Tolerance analysis for 0–1 knapsack problems

Pisinger, David; Saidi, Alima

Published in:
European Journal of Operational Research

Link to article, DOI:
10.1016/j.ejor.2016.10.054

Publication date:
2017

Document Version
Peer reviewed version

Tolerance analysis for 0-1 knapsack problems

David Pisinger[1], Alima Saidi[2]
[1] DTU Management, Technical University of Denmark,
phone: +45 45254555, email: pisinger@man.dtu.dk
[2] DIKU, University of Copenhagen

October 31, 2016

Abstract

Post-optimal analysis is the task of understanding the behavior of the solution of a problem due to changes in the data. Frequently, post-optimal analysis is as important as obtaining the optimal solution itself. Post-optimal analysis for linear programming problems is well established and widely used. However, for integer programming problems the task is much more computationally demanding, and various approaches based on branch-and-bound or cutting planes have been presented. In the present paper we study how much coefficients in the original problem can vary without changing the optimal solution vector, the so-called tolerance analysis. We show how to perform exact tolerance analysis for the 0-1 knapsack problem with integer coefficients in amortized time $O(c \log n)$ for each item, where n is the number of items, and c is the capacity of the knapsack. Amortized running times report the time used for each item, when calculating tolerance limits of all items. Exact tolerance limits are the widest possible intervals, while approximate tolerance limits may be suboptimal. We show how various upper bounds can be used to determine approximate tolerance limits in time $O(\log n)$ or $O(1)$ per item using the Dantzig bound and Dembo-Hammer bound, respectively. The running times and quality of the tolerance limits of all exact and approximate algorithms are experimentally compared, showing that all tolerance limits can be found in less than a second. The approximate bounds are of good quality for large-sized instances, while it is worth using the exact approach for smaller instances.

Keywords: Robustness & Sensitivity Analysis; Knapsack Problem; Post-optimal Analysis; Dynamic Programming

1 Introduction

In many combinatorial optimization problems the data are not given with certainty, and hence a natural question is how large the errors on the coefficients can be without distorting the sought optimal solution. Combinatorial problems, unlike linear programming problems, behave in an

unstable manner under small changes in the initial data, making tolerance analysis a challenging but important problem.

In this paper we distinguish between sensitivity analysis and tolerance analysis. Sensitivity analysis in linear programming studies in which range the coefficients can vary without changing the current basic solution. Since we do not have basic solutions in combinatorial problems, tolerance analysis studies the robustness of an optimal solution vector to perturbations in the problem coefficients. Tolerance analysis is also known as stability analysis in the literature.

Greenberg [7] gives a quite recent bibliography for post-optimal analysis in combinatorial optimization, and mentions a number of papers on knapsack problems [4, 8, 14, 22]. Klein and Holm [13] presented a general cutting-plane framework for post-optimal analysis of combinatorial problems and gave sufficient conditions for preserving the same optimal solution when the right-hand side or an objective coefficient is altered.

The *0-1 knapsack problem* consists of packing a subset of n items, each item i having a profit p_i and a weight w_i, into a knapsack of capacity c such that the overall profit is maximized. See, e.g., Kellerer *et al.* [12] for a thorough introduction. Tolerance analysis for the knapsack problem consists of determining the intervals $\alpha_{p_k} \leq p_k \leq \beta_{p_k}$ and $\alpha_{w_k} \leq w_k \leq \beta_{w_k}$ for which the profit or the weight of a given item k can be perturbed such that a given optimal solution remains optimal for the problem. Exact tolerance limits are the widest possible intervals, while approximate tolerance limits may be suboptimal (i.e., a subset of the exact tolerance limits). Notice that at any time we only alter a single item k.

Hifi *et al.* [11] proved several results that characterize the tolerance limits. Using these results they proposed two algorithms, one to compute the profit tolerances and one to compute the weight tolerances. The profit algorithm, having a running time of $O(n^2)$, applies upper bounds to derive *exact and approximate* tolerance intervals. The weight algorithm, having a running time of $O(n^2c)$, applies dynamic programming to derive *exact and approximate* tolerance intervals.

The main objective of this paper is to present an *exact* algorithm for the tolerance analysis of the 0-1 knapsack problem based on dynamic programming. The algorithm can determine the *exact* tolerance interval for the profit or weight of an arbitrary item.

This approach resembles the approach of Hifi *et al.* [11] in the way that both approaches take advantage of the dynamic programming solution, but differs in the fact that some of the results of Hifi *et al.* [11] are *approximate* while this new method is *exact* for all results. In addition the new algorithm has a better computation time, $O(nc \log n)$.

Table 1 summarizes the results of Hifi *et al.* [11] and of the present paper, reporting the time needed to compute a tolerance limit for a specific item k. The first two rows concern the perturbation of profit p_k while the next two rows concern the perturbation of weight w_k. Columns 4, 6 and 7 report running times for finding *Exact* tolerance limits, while columns 5, 8 and 9 report running times for finding *Approximate* tolerance limits. Depending on the value of the current optimal solution x_k^* the upper and lower limits can be calculated in a variety of ways. All running times are for a given item k, and it is assumed that the current optimal solution is known in advance, including the residual capacity of the solution. *Worstcase* denotes worst-case running time, while *Amortized* denotes amortized running time. Amortized running times report the time used for each item, when calculating tolerance limits of all items. Two different approximate

Table 1: Summary of the results presented by Hifi *et al.* [11] and the present paper. Notice that the quality of the approx bounds is different. The approx LP-bound generally gives the most correct tolerance limits of the three approx methods.

Perturbation	Current Solution x^*	Limit	Hifi et al. [11]		Our Results			
			Exact	Approx	Exact Worstcase	Exact Amortized	Approx LP-bound	Approx DH-bound
profit p_k	$x_k^* = 0$	α_{p_k}	$O(1)$		$O(1)$	$O(1)$		
		β_{p_k}		$O(n)$	$O(nc)$	$O(c \log n)$	$O(\log n)$	$O(1)$
	$x_k^* = 1$	α_{p_k}		$O(n)$	$O(nc)$	$O(c \log n)$	$O(\log n)$	$O(1)$
		β_{p_k}	$O(1)$		$O(1)$	$O(1)$		
weight w_k	$x_k^* = 0$	α_{w_k}		$O(n^2c)$	$O(nc)$	$O(c \log n)$	$O(\log n)$	$O(1)$
		β_{w_k}	$O(1)$		$O(1)$	$O(1)$		
	$x_k^* = 1$	α_{w_k}		$O(n^2c)$	$O(nc)$	$O(c \log n)$	$O(\log n)$	$O(1)$
		β_{w_k}	$O(n)$		$O(n)$	$O(1)$		

tolerance limits are presented in this paper using either the Dantzig upper bound (*Approx LP-bound*) or Dembo-Hammer [5] upper bound (*Approx DH-bound*).

Several related problems have been studied recently in the literature: Belgacem and Hifi [3] and [10] consider the perturbation of a subset of items in a binary knapsack problem. Monaci et al. [18] consider the related robust knapsack problem. Archetti *et al.* [1] consider the reoptimization of a knapsack problem when new items are added to the problem. Various heuristics and approximation algorithms are presented. Monaci and Pferschy [17] consider a variant of the knapsack problem where the exact weight of each item is not known in advance but belongs to a given interval. The worsening of the optimal solution is analyzed. Plateau and Plateau [21] consider how a knapsack problem can be reoptimized given that the data has been slightly modified.

The paper is organized as follows: Section 2 describes the 0-1 knapsack problem and its "dual" denoted the *weight knapsack problem*, which is advantageous when determining weight tolerance limits. Dynamic programming methods and upper/lower bounds are presented for both problems. Section 3 formally defines the tolerance analysis of a 0-1 knapsack problem and presents some special cases for which the profit or weight tolerance limits can be identified. Section 4 presents the exact profit and weight tolerance limits, and describes an $O(nc)$ algorithm per item (or $O(n^2c)$ in total) which can be used to calculate the limits. Section 5 shows how the amortized time complexity of the algorithm can be improved to $O(c \log n)$ per item (or $O(nc \log n)$ in total) by making use of overlapping subproblems in the dynamic programming. Moreover, we show how to calculate the tolerance limits by solving a single 0-1 knapsack problem. This makes it possible to use any state-of-the-art algorithm for solving the knapsack problem, and introduces the opportunity to find approximate tolerance limits by use of various upper bounds for the 0-1 knapsack problem.

2 The 0-1 Knapsack Problem

The 0-1 knapsack problem consists of packing a subset of n items into a knapsack of capacity c. Each item i has profit p_i and weight w_i and the objective is to maximize the profit of the items in the knapsack without exceeding the capacity c. Using the binary variable x_i to indicate whether item i is included in the knapsack, we get the formulation:

$$\begin{aligned} \text{(KP)} \quad \text{maximize} \quad & \sum_{i=1}^{n} p_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq c \\ & x_i \in \{0,1\}, i = 1,2,\ldots,n \end{aligned} \tag{1}$$

Without loss of generality we assume that the profits and the weights are *positive* integers (see Kellerer *et al.* [12] for transformations to this form). Also, we assume that $\sum_{i=1}^{n} w_i > c$. An optimal solution vector to KP is denoted x^* and the optimal solution value z^*. A knapsack problem with capacity c is denoted KP$[c]$, and we use the terminology KP $:=$ KP$[c]$ whenever the capacity is the original capacity. KP$[c] \setminus \{k\}$ denotes the knapsack subproblem KP$[c]$ where item k is excluded. $z(K)$ is the optimal objective function of knapsack instance K. KP(x') is the instance with variables x fixed at x', hence $z(\text{KP}(x')) = \sum_{i=1}^{n} p_i x'_i$.

The *LP-relaxed* (or fractional) knapsack problem, where $0 \leq x_i \leq 1$ for $i = 1,2,\ldots,n$ can be solved to optimality by a greedy algorithm, in which the items are sorted according to nonincreasing profit-to-weight ratio p_i/w_i and the knapsack is packed with items $1,2,\ldots$ until the first item s (the *split item*) which does not fit into the knapsack. The optimal solution value z^*_{LP} is then

$$z^*_{LP} = \sum_{i=1}^{s-1} p_i + \left(c - \sum_{i=1}^{s-1} w_i \right) \frac{p_s}{w_s}. \tag{2}$$

Knowing that all profits are integers, we may round down the solution value to $\lfloor z^*_{LP} \rfloor$ getting the *Dantzig upper bound*.

The 0-1 knapsack problem can be solved by use of dynamic programming. Let KP be a knapsack instance, and consider for $j = 0,\ldots,n$ the subproblem KP$_j[d]$ of KP consisting of the items $\{1,2,\ldots,j\}$ and having integer capacity $d \leq c$.

$$\text{KP}_j[d] = \max \left\{ \sum_{i=1}^{j} p_i x_i \,\middle|\, \sum_{i=1}^{j} w_i x_i \leq d;\ x_i \in \{0,1\}, \forall i \right\} = \text{KP}[d] \setminus \{j+1, j+2, \ldots, n\}. \tag{3}$$

Let the optimal solution value of KP$_j[d]$ be denoted $z_j(d)$. The values of $z_j(d)$ can be calculated by use of the following recursion:

$$z_j(d) = \max \Big\{ z_{j-1}(d),\ z_{j-1}(d - w_j) + p_j \Big\}, \tag{4}$$

where we set $z_0(d) = 0$ for $d = 0,\ldots,c$. We assume that $z_{j-1}(d - w_j) = -\infty$ when $d - w_j < 0$. The running time of Recursion (4) is $O(nc)$. If we only save undominated states in the dynamic programming recursion (i.e. pairs of $(d, z_j(d))$ which do not dominate each other) the running time can be limited to $O(n \min\{c, z^*\})$, where z^* is the optimal solution value of KP.

2.1 Weight Knapsack Problem

The 0-1 knapsack problem has a reverse formulation

$$\begin{aligned} \text{(WKP)} \quad \text{minimize} \quad & \sum_{i=1}^{n} w_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} p_i x_i \geq z \\ & x_i \in \{0,1\}, i = 1,2,\ldots,n \end{aligned} \tag{5}$$

where we ask for the minimum weight sum such that the profit sum z can be achieved. A specific weight knapsack problem with target sum z will be denoted WKP$[z]$. WKP$[z] \setminus \{k\}$ denotes the weight knapsack subproblem WKP$[z]$ where item k is excluded. $y(K)$ is the optimal objective function of weight knapsack instance K. WKP(x') is the instance with variables x fixed at x', hence $y(\text{WKP}(x')) = \sum_{j=1}^{n} w_j x'_j$.

If KP$[c]$ has a unique optimal solution x^* with solution value z^* then x^* will also be a unique optimal solution to WKP$[z^*]$. If several equivalent solutions to KP$[c]$ exist with the solution value z^* then WKP$[z^*]$ will return a solution using the least weight.

The *LP-relaxed* (or fractional) weight knapsack problem, where $0 \leq x_i \leq 1$ for $i = 1,2,\ldots,n$ can be solved in a similar way as the ordinary knapsack problem by sorting the items according to nonincreasing profit-to-weight ratio. Let the *weight split item* s' be the first item where the profit sum is not smaller than z. The optimal solution value z_{LP}^{w*} is then

$$z_{LP}^{w*} = \sum_{i=1}^{s'-1} w_i + \left(z - \sum_{i=1}^{s'-1} p_i \right) \frac{w_{s'}}{p_{s'}}. \tag{6}$$

Knowing that all weights are integers, we may round up the solution value to $\lceil z_{LP}^{w*} \rceil$ getting what we will call the *weight Dantzig lower bound*.

We may solve WKP by use of dynamic programming in time $O(n \min\{c, z^*\})$, where z^* is the optimal solution value of KP (see Section 3.4 in [12]).

Notice, that a weight knapsack problem WKP can be transformed to an ordinary knapsack problem KP as follows. Let the total profit and weight be given as $p^T = \sum_{i=1}^{n} p_i$ and $w^T = \sum_{i=1}^{n} w_i$. Then we have

$$\begin{aligned} y(\text{WKP}[z]) &= \min \left\{ \sum_{i=1}^{n} w_i x_i \,\middle|\, \sum_{i=1}^{n} p_i x_i \geq z; x_i \in \{0,1\}, i = 1,2,\ldots,n \right\} \\ &= w^T - \max \left\{ \sum_{i=1}^{n} w_i x_i \,\middle|\, \sum_{i=1}^{n} p_i x_i \leq p^T - z; x_i \in \{0,1\}, i = 1,2,\ldots,n \right\}. \end{aligned} \tag{7}$$

Hence we may use an ordinary KP algorithm for solving the latter maximization problem.

3 Tolerance Analysis

Let KP be a knapsack instance with an *optimal solution* x^* and an *optimal solution value* z^*. If more than one optimal solution exists, we will in the following assume that the solution x^* with the least weight sum is chosen (i.e., with the largest residual capacity). Notice that this is an important assumption since otherwise the stated theorems do not hold. Exact algorithms based on dynamic programming methods can easily be modified to satisfy the property.

Tolerance analysis for the knapsack problem consists of determining the intervals for which the profit or weight of a selected item k can be perturbed such that x^* remains an optimal (but not necessarily unique) solution.

Let $\mathrm{KP}_{\Delta p_k}$ be the knapsack instance derived from KP when a single profit p_k is substituted with $p_k + \Delta p_k$ for some $\Delta p_k \in \mathbb{Z}$.

$$\begin{array}{lll} (\mathrm{KP}_{\Delta p_k}) & \text{maximize} & \sum_{i=1}^{n} p_i x_i + \Delta p_k x_k \\ & \text{subject to} & \sum_{i=1}^{n} w_i x_i \leq c \\ & & x_i \in \{0,1\}, i = 1,2,\ldots,n \end{array} \tag{8}$$

Let $z^*_{\Delta p_k}$ be the optimal solution value to $\mathrm{KP}_{\Delta p_k}$ and $\mathrm{KP}_{\Delta p_k}(x^*) = \sum_{i=1}^{n} p_i x_i^* + \Delta p_k x_k^*$ the solution value of the original solution x^* in $\mathrm{KP}_{\Delta p_k}$. Then we define α_{p_k} and β_{p_k} to be the lower and upper, respectively, tolerance limit of p_k in KP as:

$$\alpha_{p_k} = \min_{\Delta p_k \leq 0} \{p_k + \Delta p_k \mid \mathrm{KP}_{\Delta p_k}(x^*) = z^*_{\Delta p_k}\}, \tag{9}$$

$$\beta_{p_k} = \max_{\Delta p_k \geq 0} \{p_k + \Delta p_k \mid \mathrm{KP}_{\Delta p_k}(x^*) = z^*_{\Delta p_k}\}. \tag{10}$$

Notice that since all coefficients are assumed to be integers, $\alpha_{p_k}, \beta_{p_k}$ will also be integers.

Analogously, we can define the knapsack problem $\mathrm{KP}_{\Delta w_k}$, where a single weight w_k is substituted with $w_k + \Delta w_k$ for some $\Delta w_k \in \mathbb{Z}$.

$$\begin{array}{lll} (\mathrm{KP}_{\Delta w_k}) & \text{maximize} & \sum_{i=1}^{n} p_i x_i \\ & \text{subject to} & \sum_{i=1}^{n} w_i x_i + \Delta w_k x_k \leq c \\ & & x_i \in \{0,1\}, i = 1,2,\ldots,n \end{array} \tag{11}$$

Let $z^*_{\Delta w_k}$ be the optimal solution value to $\mathrm{KP}_{\Delta w_k}$ and $\mathrm{KP}_{\Delta w_k}(x^*) = \sum_{i=1}^{n} p_i x_i^* = z^*$ be the solution value of the original solution x^* in $\mathrm{KP}_{\Delta w_k}$. Then we define α_{w_k} and β_{w_k} to be the lower and upper, respectively, tolerance limit of w_k in KP as:

$$\alpha_{w_k} = \min_{\Delta w_k \leq 0} \{w_k + \Delta w_k \mid z^* = z^*_{\Delta w_k},\ \sum_{i=1}^{n} w_i x_i^* + \Delta w_k x_k^* \leq c\}, \tag{12}$$

$$\beta_{w_k} = \max_{\Delta w_k \geq 0} \{w_k + \Delta w_k \mid z^* = z^*_{\Delta w_k},\ \sum_{i=1}^{n} w_i x_i^* + \Delta w_k x_k^* \leq c\}. \tag{13}$$

As before we notice that $\alpha_{w_k}, \beta_{w_k}$ will be integers.

The intervals $[\alpha_{p_k}, \beta_{p_k}]$ and $[\alpha_{w_k}, \beta_{w_k}]$ thus represent the tolerance intervals for p_k and w_k in KP.

The naïve way to compute the profit tolerance interval $[\alpha_{p_k}, \beta_{p_k}]$ of an item k is to decrease (or increase) by one unit the profit p_k until the given optimal solution x^* is no longer an optimal, feasible solution. The weight tolerance interval $[\alpha_{w_k}, \beta_{w_k}]$ is computed similarly. If we use the dynamic programming Recursion (4) to solve each instance of KP, the overall running time for calculating the tolerance intervals of item k becomes $O(nc(\beta_{p_k} - \alpha_{p_k} + \beta_{w_k} - \alpha_{w_k}))$. This can be slightly improved by using binary search, but still the running time may become unacceptably large.

Hifi *et al.* [11]) identified some special cases in which the tolerance intervals can be calculated easily. These results are summed up in Appendix B.

Table 2: Exact and approximate tolerance limits for an instance with $c = 9$ specified in Columns 2 and 3.

Item	Parameters and solution x^*			Exact profit intervals		Approximate profit intervals [11]		Exact weight intervals		Approximate weight intervals [11]	
i	w_i	p_i	x_i^*	α_{p_i}	β_{p_i}	α_{p_i}	β_{p_i}	α_{w_i}	β_{w_i}	α_{w_i}	β_{w_i}
1	2	6	1	4	∞	4	∞	2	2	2	2
2	3	5	0	0	6	0	4	3	∞	3	∞
3	6	8	0	0	9	0	8	5	∞	5	∞
4	7	9	1	8	∞	10	∞	5	7	7	7
5	5	6	0	0	9	0	6	5	∞	5	∞
6	9	7	0	0	15	0	15	5	∞	5	∞
7	4	3	0	0	4	0	4	2	∞	2	∞

Table 2 shows the exact profit and weight tolerance intervals computed by the naïve algorithm for a given example. It also lists the approximate tolerance intervals derived when using `ApproxLP`, the method described in [11]. The tolerance analysis guarantees that the solution remains optimal but not necessarily unique within the found interval. For item $k = 3$ we have that x^* remains optimal when $p_3 \in [0,9]$. However, for $p_3 = 9$ we have two optimal solutions: $x^* = (1,0,0,1,0,0,0)$ or $x^* = (1,0,1,0,0,0,0)$ both with $z^* = 15$.

4 Exact Tolerance Analysis

In this section we present necessary and sufficient criteria for x^* to remain optimal under various perturbations of item k. The analysis makes use of dynamic programming, where we in turn place the studied item k as the last item, in order to state the optimality criteria. An illustrative example is presented in Appendix A. Stages in the dynamic programming recursion (3) correspond to the addition of one item (i.e. one column in the dynamic programming table), while states

correspond to the individual values in the table (i.e. a capacity d and the solution $z_j(d)$). Instead of writing a state as a pair $(d, z_j(d))$, we will often use the shorthand notation $z_j(d)$ as the capacity d is implicitly given from the context.

Theorem 1 *Let KP be a knapsack instance with optimal solution x^* and let $KP_{\Delta p_k}$ be the instance where p_k is substituted with $p'_k = p_k + \Delta p_k$.*

i) if $x^*_k = 1$ then

$$x^* \text{ is optimal for } KP_{\Delta p_k} \quad \Leftrightarrow \quad p'_k \geq P \tag{14}$$

ii) if $x^*_k = 0$ then

$$x^* \text{ is optimal for } KP_{\Delta p_k} \quad \Leftrightarrow \quad 1 \leq p'_k \leq P \tag{15}$$

where

$$P = z(KP \setminus \{k\}) - z(KP[c - w_k] \setminus \{k\}).$$

Comment: The constraint $1 \leq p'_k$ in (15) is only necessary to ensure that profits are positive as we assumed in the definition of (KP). If profits are allowed to be negative, then p'_k is downward unbounded.

Proof: The main idea in the proof is to find necessary and sufficient conditions for making the same choices as in the optimal solution in a dynamic programming recursion.

Since Recursion (4) does not demand any specific ordering of the item, we may swap item k to the last position. Then the recursion says

$$z(\mathrm{KP}_{\Delta p_k}) = \max\left\{z(\mathrm{KP}_{\Delta p_k} \setminus \{k\}),\ z(\mathrm{KP}_{\Delta p_k}[c - w_k] \setminus \{k\}) + p'_k\right\}, \tag{16}$$

where the first term in the maximum expression corresponds to $x_k = 0$ and the second term corresponds to $x_k = 1$. Notice that if we choose the same term in (16) as in x^*, x_k will correspond to x^*_k and also the rest of the solution vector will be the same, since in order to find the solution vector we will backtrack from the same state in the dynamic programming recursion.

Since the only difference between KP and $\mathrm{KP}_{\Delta p_k}$ concerns element k, we have that

$$\mathrm{KP}_{\Delta p_k} \setminus \{k\} = \mathrm{KP} \setminus \{k\}, \tag{17}$$

$$\mathrm{KP}_{\Delta p_k}[c - w_k] \setminus \{k\} = \mathrm{KP}[c - w_k] \setminus \{k\}. \tag{18}$$

Recursion (16) is hence equivalent to

$$z(\mathrm{KP}_{\Delta p_k}) = \max\left\{z(\mathrm{KP} \setminus \{k\}),\ z(\mathrm{KP}[c - w_k] \setminus \{k\}) + p'_k\right\}, \tag{19}$$

$$= \max\left\{z(\mathrm{KP} \setminus \{k\}),\ z(\mathrm{KP} \setminus \{k\}) + p'_k - P\right\}. \tag{20}$$

Now, if $p'_k - P \leq 0$ the first term in the maximum expression is the largest, while if $p'_k - P \geq 0$ the second term is the largest. Since the first term corresponds to the case $x_k = 0$, and the second term corresponds to the case $x_k = 1$, the stated now follows directly. □

Theorem 2 *Let KP be a knapsack instance with optimal solution x^* and let $KP_{\Delta w_k}$ be the instance where w_k is substituted with $w'_k = w_k + \Delta w_k$.*

i) if $x^*_k = 1$ then

$$x^* \text{ is optimal for } KP_{\Delta w_k} \quad \Leftrightarrow \quad c - W \leq w'_k \leq w_k + r \tag{21}$$

ii) if $x^*_k = 0$ then

$$x^* \text{ is optimal for } KP_{\Delta w_k} \quad \Leftrightarrow \quad c - W \leq w'_k \tag{22}$$

where

$$W = \max_{0 \leq d \leq c} \left\{ d \,\middle|\, z(KP[d] \setminus \{k\}) \leq z(KP) - p_k \right\}.$$

Comment: One could have expected that w'_k was downward unbounded in (21) similarly to Theorem 1. Indeed, decreasing w'_k will make it even more attractive to choose $x_k = 1$, but if w'_k becomes too small, other items may fit into the knapsack and the optimal solution x^* will change.

Proof: Since Recursion (4) does not demand any specific ordering of the items, we may swap item k to the last position. Then the recursion says

$$z(\mathrm{KP}_{\Delta w_k}) = \max \left\{ z(\mathrm{KP}_{\Delta w_k} \setminus \{k\}),\ z(\mathrm{KP}_{\Delta w_k}[c - w'_k] \setminus \{k\}) + p_k \right\}, \tag{23}$$

where the first term in the maximum expression corresponds to $x_k = 0$ and the second term corresponds to $x_k = 1$.

Notice that in the case $x^*_k = 0$, as long as we choose the first term in (23), the whole solution vector x^* will be unchanged, since we backtrack from the same state in the dynamic programming recursion.

The situation is different in the case $x^*_k = 1$, since $z(\mathrm{KP}_{\Delta w_k}[c - w'_k] \setminus \{k\})$ refers to different states for each weight w'_k. These states may lead to different solution vectors x^* when backtracking through the dynamic programming table. Hence it is not sufficient to choose the second term in (23); we must also choose the exact same state $z(\mathrm{KP}_{\Delta w_k}[c - w_k] \setminus \{k\})$. This can only be ensured if $z(\mathrm{KP}_{\Delta w_k}[c - w'_k] \setminus \{k\}) = z(\mathrm{KP}_{\Delta w_k}[c - w_k] \setminus \{k\})$, in which case we may choose state $z(\mathrm{KP}_{\Delta w_k}[c - w'_k] \setminus \{k\})$ instead of state $z(\mathrm{KP}_{\Delta w_k}[c - w_k] \setminus \{k\})$.

Since the only difference between KP and $\mathrm{KP}_{\Delta w_k}$ concerns element k, we have that

$$\mathrm{KP}_{\Delta w_k} \setminus \{k\} \quad = \quad \mathrm{KP} \setminus \{k\}, \tag{24}$$

$$\mathrm{KP}_{\Delta w_k}[c - w'_k] \setminus \{k\} \quad = \quad \mathrm{KP}[c - w'_k] \setminus \{k\}. \tag{25}$$

This means that the Recursion (23) is equivalent to

$$z(\mathrm{KP}_{\Delta w_k}) = \max \left\{ z(\mathrm{KP} \setminus \{k\}),\ z(\mathrm{KP}[c - w'_k] \setminus \{k\}) + p_k \right\}. \tag{26}$$

In Case ii), where $x^*_k = 0$, then the solution x^* is unchanged under perturbation if and only if

$$z(\mathrm{KP} \setminus \{k\}) \geq z(\mathrm{KP}[c - w'_k] \setminus \{k\}) + p_k, \tag{27}$$

and since $x_k^* = 0$ we have $z(\text{KP} \setminus \{k\}) = z(\text{KP})$ and hence (27) is equivalent to

$$z(\text{KP}[c - w_k'] \setminus \{k\}) \leq z(\text{KP}) - p_k, \tag{28}$$

which holds exactly for $c - w_k' \leq W$.

In Case i), where $x_k^* = 1$, then the solution x^* is unchanged under perturbation if and only if

$$z(\text{KP}[c - w_k'] \setminus \{k\}) = z(\text{KP}[c - w_k] \setminus \{k\}). \tag{29}$$

We have that $z(\text{KP}[c - w_k] \setminus \{k\}) = z(\text{KP}) - p_k$ so

$$z(\text{KP}[c - w_k'] \setminus \{k\}) = z(\text{KP}) - p_k, \tag{30}$$

which holds exactly for $c - w_k' \leq W \leq w_k + r$. □

An illustrative example of Theorems 1 and 2 can be seen in Appendix A.

4.1 Profit Tolerance Limits by Solving One KP

Theorem 3 *Let KP be a knapsack instance with optimal solution x^* and let $KP_{\Delta p_k}$ be the instance where p_k is substituted with $p_k' = p_k + \Delta p_k$. The value of P in Theorem 1 can be calculated as:*

i) if $x_k^* = 1$ then $P = z(KP \setminus \{k\}) - z^* + p_k$

ii) if $x_k^* = 0$ then $P = z^* - z(KP[c - w_k] \setminus \{k\})$

Proof: We have $P = z(\text{KP} \setminus \{k\}) - z(\text{KP}[c - w_k] \setminus \{k\})$. If $x_k^* = 1$ then $z(\text{KP}[c - w_k] \setminus \{k\}) = z^* - p_k$. If $x_k^* = 0$ then $z(\text{KP} \setminus \{k\}) = z^*$. □

The theorem shows that we only need to solve one knapsack problem to calculate P.

4.2 Weight Tolerance Limits by Solving One KP

Notice that for a given limit z' we have

$$\max_{0 \leq d \leq c} \left\{ d \middle| z(\text{KP}[d]) \leq z' \right\} = \min_{0 \leq d \leq c+1} \left\{ d \middle| z(\text{KP}[d]) \geq z' + 1 \right\} - 1 = y(\text{WKP}[z' + 1]) - 1. \tag{31}$$

This means that we may calculate the tolerance limits in Theorem 2 as follows:

Theorem 4 *Let KP be a knapsack instance with optimal solution x^* and let $KP_{\Delta w_k}$ be the instance where w_k is substituted with $w_k' = w_k + \Delta w_k$. The value of W in Theorem 2 can be calculated as*

$$W = y(WKP[z^* - p_k + 1] \setminus \{k\}) - 1.$$

Proof: Since $z(\text{KP}) = z^*$ it follows from Equation (31) that

$$W = \max_{0 \leq d \leq c} \left\{ d \middle| z(\text{KP}[d] \setminus \{k\}) \leq z^* - p_k \right\} = y(\text{WKP}[z^* - p_k + 1] \setminus \{k\}) - 1.$$

□

4.3 Algorithm to Determine Tolerance Intervals

Theorems 1, 2, 3 and 4 provide us with the following tolerance limits for a given item k.

Algorithm 1 *Assume that x^* is an optimal solution to KP with solution value z^* and residual capacity r. The tolerance limits for items $k = 1,2,\ldots,n$ can then be calculated as:*

$$\text{if } x_k^* = 1 \text{ then } \begin{cases} \alpha_{p_k} = z(KP[c] \setminus \{k\}) - z^* + p_k & \beta_{p_k} = \infty \\ \alpha_{w_k} = c - y(WKP[z^* - p_k + 1] \setminus \{k\}) + 1 & \beta_{w_k} = w_k + r \end{cases}$$

$$\text{if } x_k^* = 0 \text{ then } \begin{cases} \alpha_{p_k} = 0 & \beta_{p_k} = z^* - z(KP[c - w_k] \setminus \{k\}) \\ \alpha_{w_k} = c - y(WKP[z^* - p_k + 1] \setminus \{k\}) + 1 & \beta_{w_k} = \infty \end{cases}$$

If $\alpha_{p_k} < 0$ we set $\alpha_{p_k} = 0$, and if $\alpha_{w_k} < 0$ we set $\alpha_{w_k} = 0$ to ensure nonnegative coefficients.

A possible implementation of the above algorithm is for each $k = 1,2,\ldots,n$ to remove item k from the problem and solve the remaining problem by use of Recursion (4) in time $O(nc)$. Finding all profit tolerance limits can hence be done in $O(n^2c)$. A similar approach is used for the weight tolerance limits.

5 Faster Tolerance Analysis

In the previous section we saw that the tolerance limits can be found in $O(nc)$ time for each item. This is better than the naïve algorithm presented in Section 3.

In this section we show how the time complexity can be further decreased by reusing parts of the dynamic programming table, leading to an amortized running time of $O(c \log n)$ per item. Moreover, we show how tolerance limits can be found by solving n ordinary 0-1 knapsack problems. Finally we show how approximate tolerance limits can be found in polynomial time by use of various upper bounds.

5.1 Overlapping Subproblems

If we use dynamic programming to find tolerance intervals for all items, large parts of the dynamic programming table will be the same. Indeed, our solution approach only demands that one item k is removed from the problem.

This can be exploited in a tree structure as illustrated in Fig. 1. The considered instance has 8 items, and hence we need to run dynamic programming where each of the 8 items in turn has been removed. This is shown in the last row of the figure, where each set on the form $\{1,2,3,4,5,6,7\}$ shows the order in which the items are considered in the dynamic programming Recursion (4). Higher up in the tree, we show the items which have been considered in Recursion (4). The item numbers in bold are the new items added from the above level.

In each row i of the tree we add $n/2^i$ items to each of 2^i subproblems, and there are $\lceil \log_2 n \rceil$ rows. Addition of one item by use of Recursion (4) takes $O(c)$ computation. This means that each row can be evaluated in $O(nc)$. Hence, the overall running time is $O(nc \log n)$, since we have $O(\log n)$ rows. The amortized time complexity for each item now becomes $O(c \log n)$.

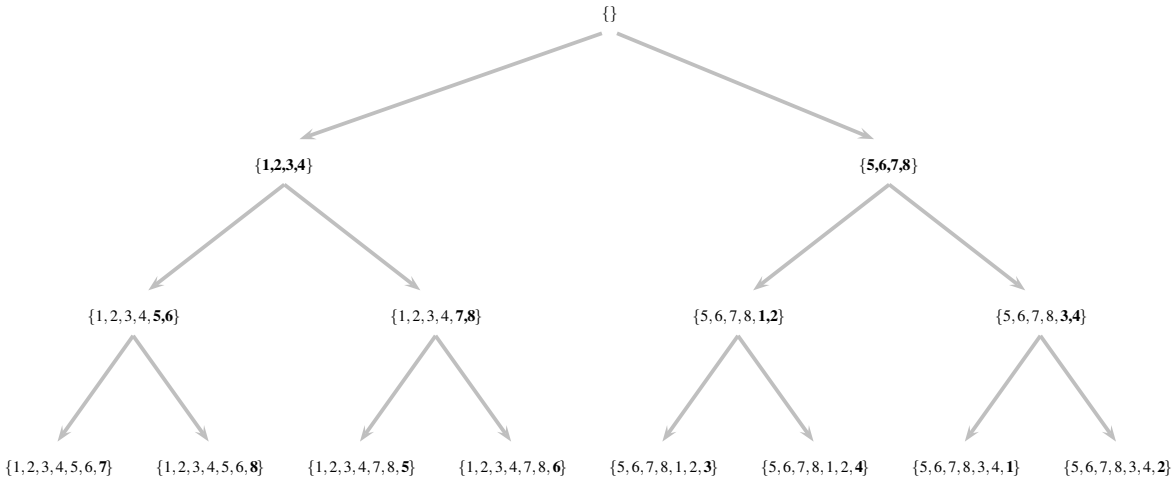

Figure 1: Tree structure which makes use of overlapping subproblems to solve n related dynamic programming problems. In each set, the item numbers in bold refers to new items.

5.2 Efficient Algorithms for Subproblems

Although the proof of Theorems 1 and 2 made use of dynamic programming, we are not limited to use this paradigm. Indeed we just need to solve the problems

$$z(\text{KP}[c] \setminus \{k\}), \quad z(\text{KP}[c-w_k] \setminus \{k\}), \quad y(\text{WKP}[z^* - p_k] \setminus \{k\}). \tag{32}$$

We can use any state-of-the-art algorithm for solving these subproblems, e.g., the algorithms presented in [15, 19]. Although these algorithms have a worst-case complexity $O(nc)$, they are much faster in practice.

5.3 Upper Bounds

Instead of solving Subproblems (32) to optimality we may use any upper bound on the solution value to obtain valid (but not necessarily optimal) tolerance limits. Let $\mathscr{U}(\text{KP})$ be an upper bound on $z(\text{KP})$ and let $\mathscr{L}(\text{WKP})$ be a lower bound on $y(\text{WKP})$. Then we have

Algorithm 2 *Approximate tolerance limits can be found as*

$$\textit{if } x_k^* = 1 \textit{ then } \begin{cases} \alpha_{p_k} = \mathscr{U}(KP[c] \setminus \{k\}) - z^* + p_k & \beta_{p_k} = \infty \\ \alpha_{w_k} = c - \mathscr{L}(WKP[z^* - p_k + 1] \setminus \{k\}) + 1 & \beta_{w_k} = w_k + r \end{cases}$$

$$\textit{if } x_k^* = 0 \textit{ then } \begin{cases} \alpha_{p_k} = 0 & \beta_{p_k} = z^* - \mathscr{U}(KP[c - w_k] \setminus \{k\}) \\ \alpha_{w_k} = c - \mathscr{L}(WKP[z^* - p_k + 1] \setminus \{k\}) + 1 & \beta_{w_k} = \infty \end{cases}$$

If the upper/lower bounds are sufficiently loose the above equations may return $\alpha_{p_k} > p_k$ or $\beta_{p_k} < p_k$. In the first case we set $\alpha_{p_k} = p_k$, and in the second case we set $\beta_{p_k} = w_k$.

Proof: To see the correctness of the above, observe that if $\mathscr{U}(\mathrm{KP}[c] \setminus \{k\}) \geq z(\mathrm{KP}[c] \setminus \{k\})$ then

$$\mathscr{U}(\mathrm{KP}[c] \setminus \{k\}) - z^* + p_k \geq z(\mathrm{KP}[c] \setminus \{k\}) - z^* + p_k = \alpha_{p_k}.$$

Moreover, if $\mathscr{U}(\mathrm{KP}[c-w_k] \setminus \{k\}) \geq z(\mathrm{KP}[c-w_k] \setminus \{k\})$ then

$$z^* - \mathscr{U}(\mathrm{KP}[c-w_k] \setminus \{k\}) \leq z^* - z(\mathrm{KP}[c-w_k] \setminus \{k\}) = \beta_{p_k}.$$

For the weight tolerance intervals we have that if $\mathscr{L}(\mathrm{WKP}[z^* - p_k + 1] \setminus \{k\}) \leq y(\mathrm{WKP}[z^* - p_k + 1] \setminus \{k\})$ then

$$c - \mathscr{L}(\mathrm{WKP}[z^* - p_k + 1] \setminus \{k\}) + 1 \geq c - y(\mathrm{WKP}[z^* - p_k + 1] \setminus \{k\}) + 1 = \alpha_{w_k}.$$

Together with Algorithm 1 we get the stated. □

Proposition 1 *If we use the Dantzig Bound in Algorithm 2 we get approximate tolerance limits in amortized time $O(\log n)$ per item.*

Proof: We first sort the items according to nonincreasing profit-to-weight ratio in time $O(n \log n)$, and then we calculate the accumulated weight sums $\overline{w}_j = \sum_{i=1}^{j} w_i$ and profit sums $\overline{p}_j = \sum_{i=1}^{j} p_i$ in overall linear time. For each item k we can then calculate a bound on $\mathrm{KP} \setminus \{k\}$ by using binary search to find the split item s as the item satisfying $\overline{w}_{s-1} \leq c < \overline{w}_s$, and then calculate the Dantzig Bound as $\overline{p}_{s-1} + (c - \overline{w}_{s-1}) p_s / w_s$. □

Proposition 2 *If we use the Dembo-Hammer upper bound [5] in Algorithm 2 we may find approximate tolerance limits in time $O(1)$ per item.*

The Dembo-Hammer upper bound is defined as follows: Let z^*_{LP} be the Dantzig upper bound given by (2), and p_s and w_s be the profit and weight of the corresponding split item s. For any perturbation Δw, the Dembo-Hammer bound is then

$$\mathscr{U}(\mathrm{KP}[c + \Delta w]) = z^*_{LP} + \Delta w \frac{p_s}{w_s}. \tag{33}$$

Notice, that the same split item s is used for all capacities, and all sub-instances of KP, making it possible to calculate the bound in $O(1)$ once s has been found. This leads to the following algorithm

Algorithm 3 *Let z^*_{LP}, p_s and w_s be defined as above in (33). The tolerance limits are then given as*

$$\textit{if } x^*_k = 1 \textit{ then } \begin{cases} \alpha_{p_k} = p_k + z^*_{LP} - z^* & \beta_{p_k} = \infty \\ \alpha_{w_k} = c + 1 - z^{w*}_{LP} - (p_k - 1) w_s / p_s & \beta_{w_k} = w_k + c - z^{w*}_{LP} - (p_k) w_s / p_s \end{cases}$$

$$\textit{if } x^*_k = 0 \textit{ then } \begin{cases} \alpha_{p_k} = 0 & \beta_{p_k} = z^* - z^*_{LP} - w_k p_s / w_s \\ \alpha_{w_k} = c + 1 - z^{w*}_{LP} - (p_k - 1) w_s / p_s & \beta_{w_k} = \infty \end{cases}$$

Proof: We have

$$\begin{aligned}
\mathscr{U}(\mathrm{KP}[c] \setminus \{k\}) &= z^*_{LP}, \\
\mathscr{U}(\mathrm{KP}[c - w_k] \setminus \{k\}) &= z^*_{LP} - w_k p_s / w_s, \\
\mathscr{L}(\mathrm{WKP}[z^* - p_k + 1] \setminus \{k\}) &= z^{w*}_{LP} - (p_k - 1) w_s / p_s, \\
\mathscr{L}(\mathrm{WKP}[z^* - p_k] \setminus \{k\}) &= z^{w*}_{LP} - (p_k) w_s / p_s.
\end{aligned}$$

Inserting the bounds in Algorithm 2 we get the stated. □

Since the Dantzig bound is tighter than the Dembo-Hammer bound we get tighter tolerance limits by using the first-mentioned bound. The time-complexity of the Dantzig bound is, however, larger.

6 Experimental Results

The object of this section is to compare the approximate method proposed by Hifi *et al.* [11] with the new exact method proposed in this paper with respect to quality and computing times. In addition we discuss the tradeoffs between the approximate method compared to the exact method.

The code used for the tests in this paper has been implemented in C/C++ and run using *gcc* on a Intel Xeon 5365 QuadCore running at 2.66 GHz. The reported running times are for a single core.

In the sequel we will use the following naming convention: Let `Naïve` be the naïve algorithm using Equations (9)–(10) and (12)–(13). Let `ExactDP` be the algorithm based on dynamic programming described in Section 5.1 which makes use of overlapping subproblems. Let `ExactKP` be the algorithm based on solving a number of knapsack and weight knapsack problems as described in Algorithm 1. Let `ApproxLP` be Algorithm 2 using the Dantzig upper bound for calculating the tolerance limits, and let `ApproxDH` be Algorithm 2 using the Dembo-Hammer upper bound for calculating the tolerance limits. In both `Naïve` and `ExactKP` we use the highly efficient `Combo` algorithm [15] for solving the individual knapsack problems to optimality.

6.1 Instance from the Literature

In this section we compare `ExactDP` with `ApproxLP` on the twenty item example presented in Hifi *et al.* [11], page 257 with capacity c equal to 420. The results have been slightly corrected as the original table contained a typo [9].

Table 3 shows the correct data for this example. The optimal solution value $z^* = 709$. The *profit* tolerance limits in Columns 5 and 6 are the approximate limits computed by `ApproxLP` described in Section 5.3 while Columns 7 and 8 show the exact intervals as computed by `ExactDP`. Columns 9 and 10 report the deviation between the two methods. A 100% deviation corresponds to the case, where `ApproxLP` found a trivial tolerance limit equal to the original profit, while `ExactDP` found a nontrivial tolerance limit.

Table 3: Example proposed by Hifi *et al.* [11] showing the *profit* and *weight* tolerance limits. The knapsack has capacity $c = 420$. For the *profits*, Columns 5 and 6 show the limits computed with `ApproxLP`. Columns 7 and 8 show the exact limits as computed by `ExactDP`. Columns 9 and 10 report the deviation in percent as $\text{dev}_\alpha = 100(\alpha_{p_k} - \alpha'_{p_k})/(p_k - \alpha'_{p_k})$ and an equivalent equation for dev_β. For the *weights*, Columns 11 to 16 show the respective values.

				Profit tolerance limits						Weight tolerance limits					
Item	Parameters		Solution	`ApproxLP`		`ExactDP`		Deviation in %		`ApproxLP`		`ExactDP`		Deviation in %	
k	p_k	w_k	x_k^*	α_{p_k}	β_{p_k}	α'_{p_k}	β'_{p_k}	dev_α	dev_β	α_{w_k}	β_{w_k}	α'_{w_k}	β'_{w_k}	dev_α	dev_β
1	80	4	1	17	∞	3	∞	18.2	0.0	4	16	2	16	100.0	0.0
2	28	3	1	16	∞	3	∞	52.0	0.0	3	15	1	15	100.0	0.0
3	54	15	1	28	∞	6	∞	45.8	0.0	15	27	13	27	100.0	0.0
4	81	25	1	37	∞	33	∞	8.3	0.0	25	37	23	37	100.0	0.0
5	31	12	1	25	∞	3	∞	78.6	0.0	12	24	10	24	100.0	0.0
6	30	17	1	30	∞	6	∞	100.0	0.0	17	29	10	29	100.0	0.0
7	39	24	1	36	∞	33	∞	50.0	0.0	24	36	22	36	100.0	0.0
8	41	27	1	39	∞	33	∞	75.0	0.0	27	39	25	39	100.0	0.0
9	68	51	1	61	∞	44	∞	70.0	0.0	51	63	49	63	100.0	0.0
10	83	65	1	74	∞	74	∞	0.0	0.0	65	77	63	77	100.0	0.0
11	33	30	1	33	∞	33	∞	0.0	0.0	30	42	28	42	100.0	0.0
12	100	91	1	92	∞	80	∞	60.0	0.0	91	103	89	103	100.0	0.0
13	74	76	0	0	74	0	74	0.0	0.0	76	∞	74	∞	100.0	0.0
14	41	44	1	41	∞	41	∞	0.0	0.0	44	56	35	56	100.0	0.0
15	47	70	0	0	57	0	67	0.0	50.0	61	∞	57	∞	30.8	0.0
16	38	69	0	0	56	0	67	0.0	37.9	52	∞	43	∞	34.6	0.0
17	32	86	0	0	74	0	74	0.0	0.0	46	∞	30	∞	28.6	0.0
18	16	62	0	0	48	0	67	0.0	37.3	30	∞	13	∞	34.7	0.0
19	6	29	0	0	15	0	30	0.0	62.5	20	∞	13	∞	43.8	0.0
20	8	40	0	0	26	0	33	0.0	28.0	22	∞	13	∞	33.3	0.0

As for the *weight* tolerance limits, the respective values are reported in Columns 11 to 16. It is interesting to see that the lower limit using `ApproxLP` very frequently just returns the original weight w_k. For the upper limits, both `ApproxLP` and `ExactDP` return the same value, since both equations only depend on the residual capacity r.

6.2 Large Instances

In this section we compare the various algorithms on large instances to examine the tradeoffs between quality and computation time. We consider four categories of randomly generated knapsack instances, which have been constructed to reflect special properties that may influence the solution process. The four instance groups are: uncorrelated, weakly correlated, strongly correlated and subset sum instances [16]. In all four groups the weights are randomly chosen as integers from a discrete uniform distribution $[1,R]$. The profits are then expressed as a function of the weights, yielding the specific properties of each group: *Uncorrelated instances*; The parameters p_j and w_j are randomly chosen as integers in $[1,R]$. *Weakly correlated instances*; Each weight w_j is chosen as a random integer in $[1,R]$ and each profit p_j is chosen as a random integer in $[w_j - R/10, w_j + R/10]$ such that $p_j \geq 1$. *Strongly correlated instances*; The weights w_j are distributed as integers in $[1,R]$ and $p_j = w_j + R/10$. *Subset sum instances*; The weights are randomly distributed integers in $[1,R]$ and $p_j = w_j$. Notice, that the profits and weights are

Table 4: Total running times in *milliseconds* on a Dell Optiflex 9020 with i7-4790 Processor (3.6 GHz). The reported times are for finding all tolerance limits, include solution of the original instance. Average values of $N = 10$ instances.

Algorithm	n	uncorrelated	weakly corr.	str. corr.	subset-sum
Naïve	5	34	13	11	9
	10	74	30	15	7
	20	186	72	46	19
	50	662	323	4656	38
	100	1791	976	72383	70
	200	6390	3251	212511	155
	500	32919	26010	3774158	342
	1000	165496	88267	19899575	634
ExactKP	5	0	0	0	0
	10	0	0	0	0
	20	0	0	1	19
	50	1	2	30	43
	100	2	4	75	72
	200	7	13	153	124
	500	26	88	613	311
	1000	108	275	1074	642
ApproxLP	5	0	0	0	0
	10	0	0	0	0
	20	0	0	0	0
	50	0	0	2	1
	100	0	1	2	1
	200	1	1	0	2
	500	4	4	6	1
	1000	14	16	15	9
ApproxDH	5	0	0	0	0
	10	0	0	0	0
	20	0	0	0	0
	50	0	0	1	0
	100	0	0	1	0
	200	0	0	0	1
	500	0	0	0	1
	1000	0	0	0	0

perturbed independently, hence the instances are not subset sum instances after perturbation.

To generate knapsack instances with the four properties, we use a knapsack generator described in Pisinger [20]. For each group we generate instances, whose sizes n vary between 5 and 1,000. For each size n we generate $N = 10$ instances. The capacity is $c = \sum w_j/2$.

We evaluate the performance of the presented algorithms on instances with relatively large profits and weights, using $R = 10,000$. For small values of R the tolerance limits frequently become trivial.

Preliminary experiments showed that the dynamic programming algorithm ExactDP with complexity $O(nc \log n)$ was not competitive with ExactKP in time $O(n^2 c)$, and ran out of space for large values of c. Hence we do not report running times for ExactDP.

Table 4 first compares the running times of the various approaches. The running times are for finding all tolerance limits, and includes the solution of the original instance. It is seen that the naïve approach Naïve is very time-consuming using several minutes for large-sized uncorrelated instances, and growing to several hours for large-sized strongly correlated instances. On the contrary, the improved exact algorithm ExactKP finds all the tolerance limits in a less than a second. The approximate algorithm ApproxLP using the Dantzig upper bound is very

Table 5: Average value of S_{p_k} (left) and average value of S_{w_k} (right).

		avg. S_{p_k}				avg. S_{w_k}			
Algorithm	n	uncorrelated	weakly corr.	str. corr.	subset-sum	uncorrelated	weakly corr.	str. corr.	subset-sum
ExactKP	5	3937.6	1133.9	1287.6	1115.3	2991.1	1361.7	850.2	751.0
	10	2885.1	696.2	427.6	55.4	2418.5	640.6	255.2	43.0
	20	3238.5	612.7	321.4	0.0	2044.7	413.5	73.9	0.0
	50	3394.2	493.5	210.7	0.0	1746.0	282.4	28.1	0.0
	100	3175.8	492.1	349.6	0.0	1407.0	254.3	23.7	0.0
	200	3176.6	512.1	607.5	0.0	1285.8	254.3	98.9	0.0
	500	3107.4	501.1	216.8	0.0	1248.7	247.2	8.1	0.0
	1000	3104.4	506.1	437.0	0.0	1210.1	252.7	38.8	0.0
ApproxLP	5	1710.3	6.5	16.1	0.0	1797.7	528.2	242.1	169.6
	10	1529.6	117.6	40.3	0.0	1583.8	175.2	82.6	7.7
	20	2016.4	152.4	59.4	0.0	1439.7	177.8	17.5	0.0
	50	2603.2	270.0	53.2	0.0	1116.8	157.6	5.0	0.0
	100	2711.3	353.7	98.9	0.0	1118.2	179.7	4.1	0.0
	200	2898.8	410.2	205.6	0.0	1137.7	202.4	18.4	0.0
	500	2979.6	460.5	51.3	0.0	1173.1	226.0	1.0	0.0
	1000	3026.4	476.9	129.1	0.0	1169.3	235.7	5.9	0.0
ApproxDH	5	177.6	5.9	0.0	0.0	288.1	95.2	4.1	24.4
	10	271.5	67.1	3.5	0.0	353.4	81.5	3.0	0.0
	20	690.4	80.3	2.3	0.0	806.9	90.7	2.0	0.0
	50	749.4	128.1	3.2	0.0	982.7	132.2	2.8	0.0
	100	757.2	156.4	4.0	0.0	976.0	157.9	3.6	0.0
	200	845.9	189.5	18.3	0.0	1073.3	190.6	16.1	0.0
	500	865.2	220.6	1.1	0.0	1137.1	220.8	1.0	0.0
	1000	868.5	232.3	6.6	0.0	1155.5	232.5	5.8	0.0

Table 6: Number of items where $S_{p_k} > 0$ (left) and number of items where $S_{w_k} > 0$ (right).

		$S_{p_k} > 0$				$S_{w_k} > 0$			
Algorithm	n	uncorrelated	weakly corr.	str. corr.	subset-sum	uncorrelated	weakly corr.	str. corr.	subset-sum
ExactKP	5	5	5	5	5	5	5	5	5
	10	10	10	10	10	10	10	10	10
	20	20	20	19	0	20	20	20	0
	50	50	50	22	0	50	50	8	0
	100	100	100	47	0	100	100	14	0
	200	200	200	133	0	200	200	60	0
	500	500	499	117	0	500	500	9	0
	1000	1000	999	444	0	1000	950	65	0
ApproxLP	5	3	0	0	0	4	3	3	2
	10	6	4	2	0	8	7	6	5
	20	16	10	5	0	18	16	11	0
	50	44	34	10	0	47	42	5	0
	100	91	83	35	0	96	91	10	0
	200	188	180	121	0	194	189	55	0
	500	485	475	108	0	491	462	8	0
	1000	977	974	437	0	989	886	62	0
ApproxDH	5	0	0	0	0	0	0	0	0
	10	1	2	0	0	1	2	0	0
	20	6	5	0	0	6	5	0	0
	50	15	16	1	0	16	17	1	0
	100	32	39	3	0	33	40	3	0
	200	69	87	26	0	69	87	26	0
	500	178	232	7	0	178	232	7	0
	1000	360	483	62	0	360	483	62	0

fast, using no more than 0.02 seconds for all large-scaled instances. The fastest algorithm is the `ApproxDH` which only spends a few milliseconds to find the tolerance limits.

The size of a tolerance interval for p_k and w_k is $S_{p_k} = (\beta_{p_k} - \alpha_{p_k})$ and $S_{w_k} = (\beta_{w_k} - \alpha_{w_k})$, respectively. Table 5 reports the average value of S_{p_k} and S_{w_k}. It is seen that the tolerance limits decrease for increasing correlation. For the most correlated instances, the subset-sum instances, the tolerance limits are zero except for very small instances. The range of the tolerance intervals for the weights are significantly smaller then the tolerance intervals for the profits. For the strongly correlated instances, the average lower tolerance limits are less then 2% on average.

Comparing the values of the tolerance limits in Table 5, it is seen that `ApproxLP` using the Dantzig upper bound is close to the exact tolerance limits for large-sized uncorrelated, weakly correlated and subset-sum instances. However, for small-sized instances the use of the Dantzig upper bound significantly decreases the tolerance limits. The `ApproxDH` algorithm using the Dembo-Hammer upper bound finds considerably smaller tolerance limits than the previous approaches for the profits, and hence is hardly of any use. However for the weights `ApproxDH` finds comparable tolerance limits.

Finally, Table 6 reports the number of items with non-zero tolerance limits. It is seen that for the uncorrelated and weakly correlated instances nearly all items have a non-trivial exact tolerance limit for the profits and the weights. For the strongly correlated instance around half of the items have a non-zero tolerance limit for the profits, and considerably fewer for the weights. For the subset-sum instance, non-trivial tolerance limits are found only for small instances. This may be due to the fact that for larger instances numerous solutions exist with objective value equal to the capacity c, and hence a small increase in the profit or weight of an item will change the solution.

7 Conclusion

Tolerance analysis makes it possible to test the robustness of a model, and to fix model input that has no effect on the output. Knapsack problems frequently occur as subproblems in more complex combinatorial algorithms, e.g., as pricing problem (Vanderbeck [23]) or for cut separation (Balas [2], Fukasawa and Goycoolea [6]). Knowing the tolerance limits can perhaps be used to derive tighter cuts, and it can save time in solving pricing problems when the parameters only have been changed within the tolerance limits.

We have presented an algorithm for finding the exact tolerance limits in amortized time $O(c \log n)$ for each item. Moreover we have showed that the tolerance limits can be found by solving a single knapsack problem, making it possible to use all current knapsack solvers and upper bound algorithms for calculating the limits.

Notice, that although we used dynamic programming (or actually a recursive formulation) to prove the theorems, the found tolerance limits are independent of dynamic programming. This means that the theorems, with a few modifications, also hold if the profits and weights are rational numbers.

The computation experiments show that the exact algorithm `ExactKP` is performing very well, making it possible to find all tolerance limits in roughly one second for large-sized in-

stances. If faster solution times are needed, one may use the `ApproxLP` algorithm for larger instances (say, $n > 100$) and `ExactKP` for the smaller instances. This combination results in very fast running times, and a good quality of the tolerance limits.

Future work should focus on how the presented results can be generalized to other problems that can be solved in pseudo-polynomial time. It should be possible to use similar arguments from dynamic programming to analyze the tolerance limits.

References

[1] C. Archetti, L. Bertazzi, and M.G. Speranza. Reoptimizing the 0-1 knapsack problem. *Discrete Applied Mathematics*, 158(17):1879–1887, 2010.

[2] E. Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8:146–164, 1975.

[3] T. Belgacem and M. Hifi. Sensitivity analysis of the optimum to perturbation of the profit of a subset of items in the binary knapsack problem. *Discrete Optimization*, 5:755–761, 2008.

[4] R.E. Burkard and U. Pferschy. The inverse-parametric knapsack problem. *European Journal of Operational Research*, 83:376–393, 1995.

[5] R.S. Dembo and P.L. Hammer. A reduction algorithm for knapsack problems. *Methods of Operations Research*, 36:49–60, 1980.

[6] R. Fukasawa and M. Goycoolea. On the exact separation of mixed integer knapsack cuts. *Math. Program., Ser. A*, 128:19–41, 2011.

[7] H.J. Greenberg. An annotated bibliography for post-solution analysis in mixed integer programming and combinatorial optimization. In D.L. Woodruff, editor, *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search*, pages 97–148. Kluwer, 1998.

[8] P. Hansen and J. Ryan. Testing integer knapsacks for feasibility. *European Journal of Operational Research*, 88:578–582, 1996.

[9] M. Hifi, 2007. Personal correspondence.

[10] M. Hifi and H. Mhalla. Sensitivity analysis to perturbations of the weight of a subset of items: The single knapsack case study. *Electronic Notes in Discrete Mathematics*, 36:439–446, 2010.

[11] M. Hifi, H. Mhalla, and S. Sadfi. Sensitivity of the optimum to pertubations of the profit or weight of an item in the binary knapsack problem. *Journal of Combinatorial Optimization*, 10:239–260, 2005.

[12] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[13] D. Klein and S. Holm. Integer programming post-optimal analysis with cutting planes. *Management Science*, 25:64–72, 1979.

[14] L. N. Kozeratskaya, T. T. Lebedeva, and I. V. Sergienko. Stability, parametric, and postoptimality analysis of discrete optimization problems. *Cybernetics and Systems Analysis*, 19:522–535, 1983.

[15] S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45:414–424, 1999.

[16] S. Martello and P. Toth. An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research*, 1:169–175, 1977.

[17] M. Monaci and U. Pferschy. On the robust knapsack problem. *SIAM Journal on Optimization*, 23(4):1956–1982, 2013.

[18] M. Monaci, U. Pferschy, and P. Serafini. Exact solution of the robust knapsack problem. *Computers & OR*, 40:2625–2631, 2013.

[19] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45:758–767, 1997.

[20] D. Pisinger. Core problems in knapsack algorithms. *Operations Research*, 47:570–575, 1999.

[21] A. Plateau and G. Plateau. Reoptimization for 0-1 knapsack problems. In *Proc. Congreso Latino-Iberoamericano de Investigacion Operativa*, Rio de Janeiro, Brazil, September 24–28 2012.

[22] J. Seelander. Einige bemerkungen zur bestimmung von stabilitetsbereichen in der reinganzzahligen linearen optimierung. *Math. Operationsforsch. Statistic, Ser. Optimization*, 11:261–271, 1980.

[23] F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Math. Program., Ser. A*, 86:565–594, 1999.

Appendix A - Illustration of the Main Theorems

Fig. 2 shows two knapsack instances and the corresponding dynamic programming tables $z_j(d)$. Both instances have $n = 4$ and $c = 7$. We want to find the tolerance limits for the last item $k = 4$.

In the **left** instance we find $z = 8, r = 0$ hence $P = 3, W = 6$. Since $x_4^* = 0$, Theorems 1 and 2 give us the tolerance limits $0 \leq p_4' \leq 3$ and $1 \leq w_4'$.

We could also reach these limits from the dynamic programming. For p_4' we note that $z_4 = \max\{z_3(c), z_3(c - w_4) + p_4'\} = \max\{8, 5 + p_4'\}$, where the first term is chosen as long as $p_4' \leq 3$.

For w_4' we note that $z_4 = \max\{z_3(c), z_3(c - w_4') + p_4\} = \max\{8, z_3(c - w_4') + 1\}$, where the first term is chosen as long as $z_3(c - w_4') \leq 7$, which by inspection in $z_3(d)$ can be seen to hold for $c - w_4' \leq 6$.

In the **right** instance we find $z = 11, r = 3$ and $P = 3, W = 5$. Since $x_4^* = 1$, Theorems 1 and 2 give us the tolerance limits $p_4' \geq 3$ and $2 \leq w_4' \leq 5$.

Using dynamic programming we note for p_4' that $z_4 = \max\{z_3(c), z_3(c - w_4) + p_4'\} = \max\{8, 5 + p_4'\}$, where the second term is chosen as long as $p_4' \geq 3$.

For w_4' we note that $z_4 = \max\{z_3(c), z_3(c - w_4') + p_4\} = \max\{8, z_3(c - w_4') + 6\}$, where the second term is chosen as long as $z_3(c - w_4') \geq 2$. However, this is not sufficient to ensure that the current optimal solution $x^* = (1, 0, 0, 1)$ remains optimal, since choosing e.g., $c - w_4' = 6$ (i.e., $w_4' = 1$) will result in an optimal solution $x^* = (0, 1, 0, 1)$ with value $z = 13$. As observed in the proof of Theorem 2, the current optimal solution $x^* = (1, 0, 0, 1)$ remains optimal if and only if $z_3(c - w_4') = z_3(c - w_4) = 5$, which by inspection in $z_3(d)$ can be seen to hold for $2 \leq c - w_4' \leq 5$.

j	1	2	3	4
p_j	5	7	3	1
w_j	2	6	5	2

$d \backslash j$	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	5	5	5	5
3	0	5	5	5	5
4	0	5	5	5	6
5	0	5	5	5	6
6	0	5	7	7	7
7	0	5	7	8	8

j	1	2	3	4
p_j	5	7	3	6
w_j	2	6	5	2

$d \backslash j$	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	5	5	5	6
3	0	5	5	5	6
4	0	5	5	5	11
5	0	5	5	5	11
6	0	5	7	7	11
7	0	5	7	8	11

Figure 2: Two knapsack instances and the corresponding dynamic programming tables $z_j(d)$. In both instances $n = 4$ and $c = 7$. The two instances only differ with respect to p_4. In the left instance, $z^* = 8$ and $x^* = (1, 0, 1, 0)$. In the right instance, $z^* = 11$ and $x^* = (1, 0, 0, 1)$.

Appendix B - Special Cases

In this section we consider some special cases in which the tolerance intervals can be identified easily as shown by Hifi *et al.* [11]). First, we need some definitions: We define the *residual capacity* $r \geq 0$ of a KP as

$$r = c - y(\text{WKP}[z^*]), \tag{34}$$

where z^* is the optimal solution value to KP. If several solutions to KP have the same solution value z^* the residual capacity r is the largest possible free space among all optimal solutions.

Lemma 1 *(Theorem 2.1 in Hifi et al. [11]) If x^* is an optimal solution for KP, and ($\Delta p_k \geq 0$ and $x_k^* = 1$) or ($\Delta p_k \leq 0$ and $x_k^* = 0$), then x^* is an optimal solution for $KP_{\Delta p_k}$.*

Lemma 1 states that x^* remains optimal for $\text{KP}_{\Delta p_k}$ if $x_k^* = 1$ and p_k increases, or if $x_k^* = 0$ and p_k decreases. The tolerance limits are hence upward (downward) unlimited in these cases as long as the profit $p_k + \Delta p_k$ remains nonnegative (since we have defined the KP as having nonnegative profits.)

Lemma 2 *(Theorem 3.1 in Hifi et al. [11]) If x^* is an optimal solution for KP, and ($0 \leq \Delta w_k \leq r$ and $x_k^* = 1$) or ($\Delta w_k \geq 0$ and $x_k^* = 0$), then x^* is an optimal solution for $KP_{\Delta w_k}$.*

Lemma 2 states that x^* remains optimal for $\text{KP}_{\Delta w_k}$ if $x_k^* = 0$ and the weight is increased. Moreover if $x_k^* = 1$ and the weight is increased up to the residual capacity, the solution x^* remains optimal.